Cross-Layer Resilience Exploration

Eric Cheng, Hyungmin Cho, Subhasish Mitra
Department of EE and Department of CS
Stanford University
Stanford, CA, USA, 94305

Lukasz G. Szafaryn, Kevin Skadron, Mircea Stan
Department of ECE and Department of CS
University of Virginia
Charlottesville, VA, USA, 22904

Chen-Yong Cher
Computer Architecture Department
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA, 10598

Shahrzad Mirkhani, Jacob A. Abraham
Department of ECE
University of Texas at Austin
Austin, TX, USA, 78712

Abstract: *Low-cost fault tolerance requires careful combination of fault tolerance techniques across all levels of the system stack. We describe a systematic framework, applicable to simple and complex cores as well as specialized accelerators, to explore this cross-layer design space in terms of area, power, and performance costs, and present illustrative results.*

Keywords: cross-layer resilience; fault-tolerance; reliability; soft errors

Introduction

It is clear that at recent technology nodes, and moving forward, even commodity systems are increasingly susceptible to hardware failures. We show the benefits of combining techniques across layers, from circuit-level to application-level, which allows the designer to select a desired fault tolerance with much lower overhead. We present a systematic methodology and a new framework that allows for quick exploration and analysis of cross-layer resilience. We demonstrate the effectiveness of our framework using a wide range of resilience techniques: from circuit-, logic-, and architecture-level techniques to software-implemented and application-specific resilience.

Related Work

There has been significant research into various protection schemes across all system layers that help improve overall system reliability. The challenge is determining how to combine and integrate these techniques. Prior work on fault tolerance has typically proposed techniques that target a single layer of the system stack. Even prior work showing how to combine prior techniques to reduce overhead (e.g., [14]) still operates within a single layer of the system stack.

Unfortunately, very little literature exists on such cross-layer tradeoffs. Some prior work (e.g., [2, 4]) has suggested the need for cross-layer integration of resilience techniques, but ours is the first truly cross-layer framework to systematically evaluate area, power, and performance overheads under various reliability constraints.

Methodology and Framework

A fundamental difference of our methodology to past practices is that we apply a systematic approach to cross-layer resilience based on in-depth analysis rather than designer intuition to guide cross-layer combinations. Furthermore, our new framework (Fig. 1) allows us to quickly explore, evaluate, and optimize cross-layer resilience. The main components of the framework are a very rich resilience library that incorporates resilience techniques from across the system stack, a carefully crafted cross-layer fault injector that is highly accurate and fast, a system-aware design-space exploration that allows for quick exploration of the hundreds of possible design points, and a layout-aware exploration that accounts for all wire and routing considerations, which impact overheads significantly, for key designs. Physical design overheads are evaluated using a 28nm TSMC library and Synopsys design tools (Design Compiler, IC compiler, and PrimeTime) to perform synthesis, place-and-route, and power and timing analysis. The framework automatically and intelligently inserts resilience into baseline designs to provide optimized cross-layer solutions and generates reports detailing the achieved reliability as well as the area, power, and performance overheads of the resilient design.

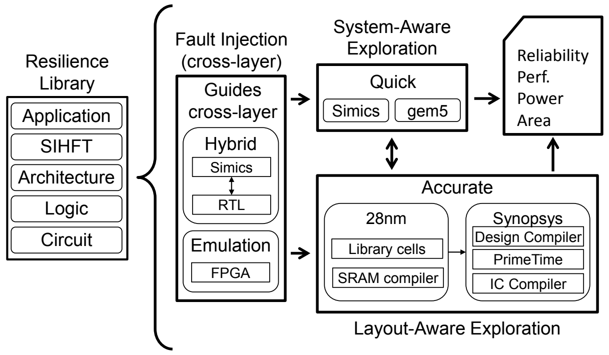


Figure 1. Our cross-layer framework

Case Study

We present, as a case study, the application of our cross-layer methodology and framework on the Leon3. For the purposes of this study, we focus on the particular case of single event upsets (SEU) caused by soft errors (radiation-induced transient errors) since the transient nature of these types of errors lead to interesting challenges such as architectural masking factors in the design.

Although we only present results for Leon3 (a simple, in-order core), we have also applied our framework to IVM (a complex, out-of-order core) to reinforce the fact that our methodology and framework is applicable to any system (ranging from simple embedded systems to complex server-class systems) and any arbitrary fault model (permanent, transient, multi-bit, etc.)

System Design Analysis

Using flip-flop-level fault injection, we rank the vulnerability of each flip-flop in the processor in terms of its likelihood to propagate faults [3]. This allows the designer to target a desired level of fault tolerance, which guides cross-layer exploration. Analysis is conducted on a diverse set of benchmarks including the SPECINT 2000 and DARPA PERFECT benchmarks. We evaluate fully routed designs in order to accurately account for crucial physical design aspects like wire routing.

Resilience Technique Characterization

Here, we briefly summarize some of the resilience techniques we have explored and illustrate how our characterization affects their application in our study.

Circuit-level Techniques: At the circuit-level, we look at *radiation hardened flip-flops*, which are flip-flops designed to uphold the bit representation of their output circuit even under particle strikes [1, 6, 10, 13]. We look at a range of flip-flops that offer tradeoffs in area and power versus the amount of resilience provided (Table 1).

Table 1. Hardened flip-flop comparison

Type	Soft Error Rate	Area	Power	Delay
Baseline	1.0	1.0	1.0	1.0
Light	2.5×10^{-1}	1.2	1.1	1.2
Moderate	5.0×10^{-2}	1.3	1.5	1.7
Heavy	2.0×10^{-4}	2.0	1.8	1.0

Logic-level Techniques: At the logic-level, we explore *logic parity*, a technique which detects soft-errors affecting groups of sequential cell elements [9]. Naïve implementations will result in significant impact on clock frequency. In contrast, by implementing pipelined parity (accomplished by adding extra pipeline flip-flops), we maintain the original design frequency (Fig. 2).

However, one must still exercise extreme care since there are many different parameters to use for optimization, such as parity group size, flip-flop vulnerability, cell locality, and timing slack. Choosing the wrong optimization parameter can lead to 40% area and 80% power overhead as compared to the best heuristic. Our framework searches the design space to find the best overall solution.

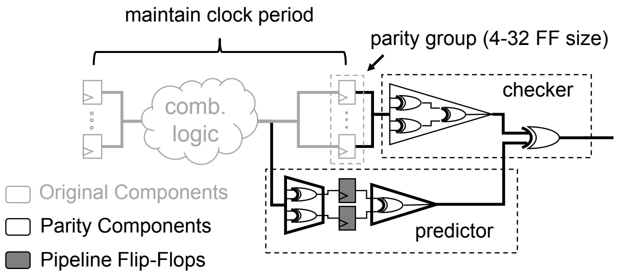


Figure 2. Pipelined logic parity

Architecture-level Techniques: Most micro-architectural techniques have high cost in area (e.g., redundancy) or performance (e.g., redundant multi-threading). One promising technique that we evaluate is *Dataflow Checking* (*DFC*) [8], which detects errors in control flow as well as changes in operations and operands. Hardware overhead for detection is negligible, but recovery requires buffering results from an entire basic block, making the overhead prohibitive in a simple processor such as Leon3. In IVM, however, the reorder buffer allows for low-cost recovery; but DFC covers a smaller fraction of the design. Quantification of these tradeoffs is work in progress.

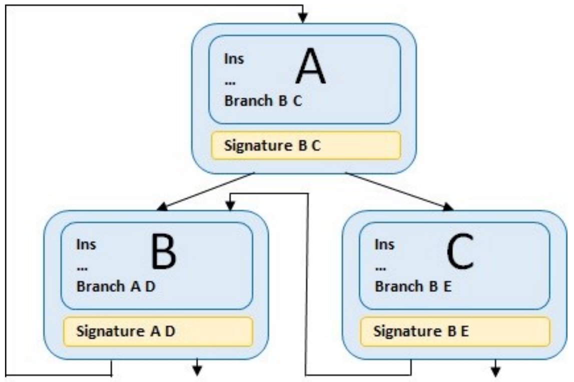


Figure 3. Transitions between basic blocks in DFC

Software-level Techniques: We analyze two Software-Implemented Hardware Fault Tolerance (SIHFT) resilience techniques*: Control-Flow Checking by Software Signatures* (*CFCSS*) [11] and *Error Detection by Duplicated Instructions* (*EDDI*) [12]. Although these techniques do not incur area and power overhead from additional hardware, we see an application slowdown of on average 1.5× for CFCSS and 2.2× for EDDI. Furthermore, these software techniques tend to cover only a limited range of flip-flops (around 20%).

Application-specific Techniques: We look at *Algorithm Based Fault Tolerance* (*ABFT*) applied to select PERFECT benchmarks [5]. We have found that these techniques generally lie in the ideal region of high error-detection rate at low runtime overhead. They only provide protection when running the corresponding algorithm, so in general-purpose processors, hardware protection is still needed. However, in accelerators, ABFT can eliminate the need for other fault-tolerance mechanisms in dedicated hardware, or in more general-purpose processors, allow hardware protection to be turned off for potential power savings when the protected algorithms are running. We note that, some ABFT techniques, such as for FFT, only detect but cannot correct errors.

Recovery Considerations: We round out our analysis by examining recovery options for Leon3. The simplest cross-layer recovery method takes advantage of the built-in pipeline-flush mechanism (Fig. 4). Errors that occur and are detected before reaching the memory write stage of the pipeline can be flushed and re-executed. Implementing this recovery mechanism costs 0.6% area and 0.9% power. However, it does require that the last stages in the pipeline be protected using techniques that both detect and correct (e.g., hardened flip-flops). The second method of recovery we evaluate is a hardware recovery unit (R-Unit), which utilizes instruction checkpointing [7] (Fig. 5). Although more general, this unit is relatively expensive on the Leon3, requiring 16% area and 21% power overhead.

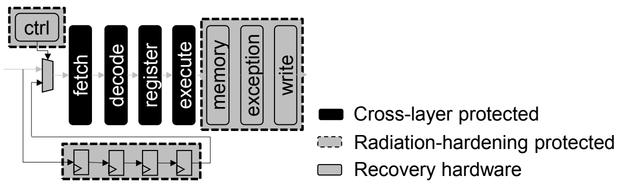


Figure 4. Cross-layer (flush) recovery

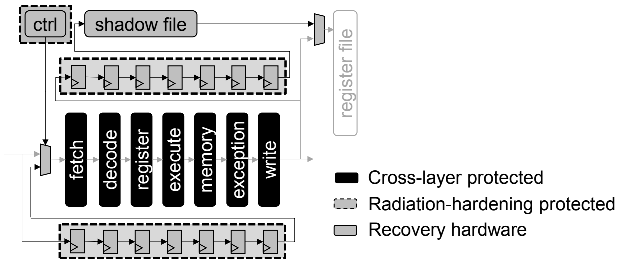


Figure 5. R-unit recovery

Cross-Layer Combinations

We examine case studies on the Leon3 core to demonstrate systematic cross-layer exploration using our framework.

Circuit- and Logic-level Combinations: We show that careful combination of hardening and parity protection for flip-flops, allows designs that achieve significant savings as compared to a single-layer (hardening-only or parity-only) approach. For instance, in Leon3, even after considering recovery, our cross-layer solution yields a savings of anywhere from 1.1-1.9× in area and 1.1-1.5× in power (after place and route) as compared to the best single-layer resilience approach. In fact, for all soft-error rate (SER) improvement targets (from 5-5,000× improvement), our framework automatically generates a lower cost solution than the best single-layer solution available (Fig. 6).

Circuit-, Logic-, and Architecture-level Combinations: Using our two-layer solution as a baseline, we investigate whether incorporating additional layers into our cross-layer exploration yields further savings. We look at Data Flow Checking (DFC), an architecture-level technique. For Leon3, DFC detects errors in flip-flops carrying program counter and instruction content throughout the pipeline until the last stage (write-back) where the check is performed. However, these flip-flops lack protection during basic blocks following branches that are indirect or outside of the binary address range (~20% of duration). Moreover, some flip-flops only have errors that are occasionally detected by DFC, resulting in partial detection. Overall, 35% of flip-flops are fully or partially protected with a 50% average detection rate.

As a result, the low per-flip-flop detection requires significant low-level (hardening and parity) protection, which negates the benefits of using DFC. Moreover, coupled with the high design cost due to compiler modifications and the corresponding hardware checker, high area cost of the checker hardware, and the high (~20%) performance overheads, we find that cross-layer solutions incorporating this specific architecture-level technique are not viable solutions.

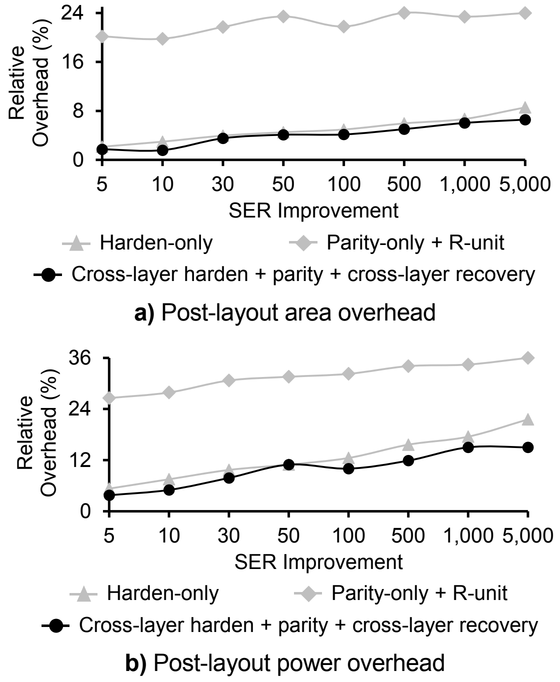
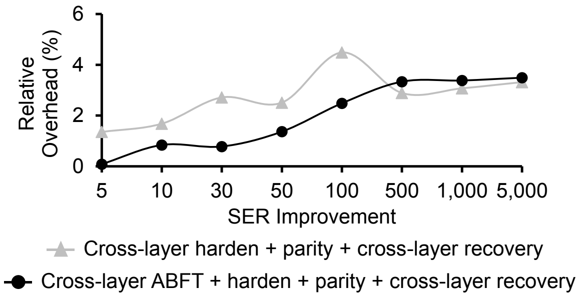


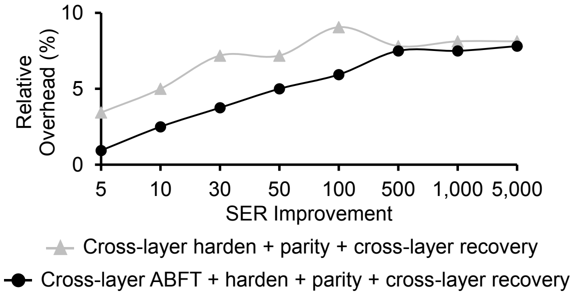
Figure 6. Post-layout overhead comparison (circuit- and logic-level resilience)

Circuit-, Logic-, and Application-specific Combinations: Let us consider the classical case of adding Algorithm Based Fault Tolerance (ABFT) correction for a processor dedicated to inner product. We find that a cross-layer combination actually yields anywhere from a 1.8-3.5× area improvement (for a dedicated processor) and 1.4-3.7× power improvement (post-layout) for resilience improvements in the range of 5-100× as compared to our previous best two-layer approach (Fig. 7). Keep in mind that these improvements are even higher when we compare to the best single-layer approach instead.

These significant benefits are achievable because ABFT is actually allowing us to reduce the amount of low-level hardware protection required (by 20-50%) to meet our resilience targets However, for highly resilient design targets (e.g., 500-5,000×), the addition of ABFT into our cross-layer analysis yields negligible benefits. Although ABFT has high coverage for a small number of highly vulnerable flip-flops, residual errors of non-fully protected flip-flops will still require low-level hardware augmentation in order to achieve a highly-resilient design.



a) Post-layout area overhead



b) Post-layout power overhead

Figure 7. Post-layout overhead comparison (circuit-, logic-, and application-specific resilience)

Circuit-, Logic-, Architecture-, and Application-specific Combinations: Although we know that cross-layer combinations of DFC (an architecture-level technique) and low-level hardware resilience (hardening and parity) do not yield any meaningful benefits, we want to discover if this still holds when we incorporate application-specific techniques due to the fact that these two higher-level techniques could potentially complement one another.

Unfortunately, although DFC provides some minimal benefits (i.e., the number of flip-flops requiring low-level hardware augmentation drops by an additional 10-20%), the area and power costs of the DFC checker itself negate any benefits afforded. In actuality, this cross-layer combination could be significantly worse, because DFC (in the case of an embedded, real-time system) will incur tremendous recovery costs.

Conclusions

We present a new methodology and framework for exploring, evaluating, and optimizing cross-layer resilience. Our methodology and framework is applicable to any arbitrary design and for any arbitrary use case. By taking a system-level view of resilience, we can systematically find *effective* cross-layer combinations to achieve desired resilience at much lower cost than the best possible single-layer solutions.

Acknowledgements

This work is sponsored in part by Defense Advanced Research Projects Agency, Microsystems Technology Office (MTO), under contract no. HR0011-13-C-0022. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. This document is: Approved for Public Release, Distribution Unlimited.

References

1. Calin, T., et al., "Upset Hardened Memory Design for Submicron CMOS Technology," IEEE Trans. on Nuclear Science, vol. 43, no. 6, pp. 2874-2878, 1996.
2. Carter, N., H. Naeimi, and D. Gardner, “Design Techniques for Cross-layer Resilience,” Proc. Design and Test Europe, pp. 1023-1028, 2010.
3. Cho, H., et al., "Quantitative Evaluation of Soft Error Injection Techniques for Robust System Design," Proc. Design Automation Conference, pp. 1-10, 2013.
4. DeHon, A., H. Quinn, and N. Carter, “Vision for Cross-layer Optimization to Address the Dual Challenges of Energy and Reliability,” Proc. Design and Test in Europe, pp. 1017-1022, 2010.
5. Huang, K, and J. Abraham, “Algorithm-Based Fault Tolerance for Matrix Operations,” IEEE Trans. Computers, vol. C-33, no. 6, pp. 518-528, 1984.
6. Lee, H., et al., "LEAP: Layout Design through Error-Aware Transistor Positioning for Soft-Error Resilient Sequential Cell Design," Proc. International Reliability Physics Symposium, pp. 203-212, 2010.
7. Meaney, P., et al., "IBM z990 Soft Error Detection and Recovery," IEEE Trans. on Device and Materials Reliability, pp. 419-427, 2005.
8. Meixner, A., et al., "Argus: Low-Cost, Comprehensive Error Detection in Simple Cores," 40th Annual IEEE/ACM International Symposium on Microarchitecture, pp.210-222, 2007.
9. Mitra, S. and E.J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?" Proc. International Test Conference, pp. 985-994, 2000.
10. Mitra, S., et al., “Robust System Design with Built-In Soft Error Resilience,” IEEE Computer, vol. 38, no. 2, pp. 43-52, 2005.
11. Oh, N., P. Shirvani, and E.J. McCluskey, “Control-Flow Checking by Software Signatures,” IEEE Trans. Reliability, vol. 51, no. 1, pp. 111-122, 2002.
12. Oh, N., P. Shirvani, and E.J. McCluskey, “Error Detection by Duplicated Instructions in Super-scalar Processors,” IEEE Trans. Reliability, vol. 51, no. 1, pp. 63-75, 2002.
13. Rodbell, K., et al., "32 and 45 nm Radiation-Hardened-by-Design (RHBD) SOI Latches," IEEE Trans. on Nuclear Science, vol. 58, no. 6, pp. 2702-2710, 2011.
14. Szafaryn, L. et al., "Evaluating Overheads of Multibit Soft-Error Protection in the Processor Core," IEEE Micro, vol. 33, no. 4, pp. 56-65, 2013.